Rectangular Selection of Components in Large 3D Models on the Web

Sebastian Friston
University College London
London, UK

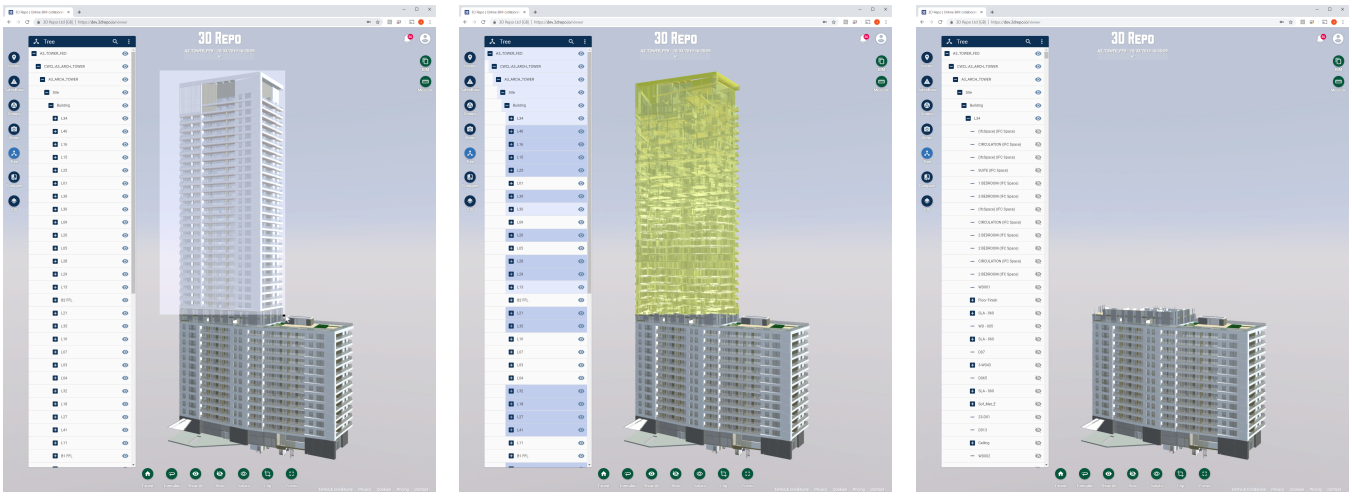
Jozef Doboš
3D Repo Ltd
London, UK

Charence Wong
3D Repo Ltd
London, UK

Carmen Fan
3D Repo Ltd
London, UK

Santiago Montero
3D Repo Ltd
London, UK

Anthony Steed
University College London
London, UK



(a) (b) (c)

Figure 1: User-driven rectangular selection on a model with 75k components within the 3D Repo cloud platform. a) The user draws a rectangle to indicate the desired selection volume. b) The system performs intersection calculations in real-time and displays the selected objects with an "X-ray" highlighting. c) The user hides the selection and the state change is reflected in the component tree. Model (A3) courtesy of Canary Wharf Contractors.

ABSTRACT

We introduce a novel method for rectangular selection of components in large 3D models on the web. Our technique provides an easy to use solution that is developed for renderers with partial fragment shader support such as embedded systems running WebGL. This method was implemented using the Unity 3D game engine within the 3D Repo open source framework running on a web browser. A case study with industrial 3D models of varying complexity and object count shows that such a solution performs within reasonable rendering expectations even on underpowered devices without a dedicated graphics card.

CCS CONCEPTS

• **Computing methodologies** → *Rasterization*; *Image processing*.

KEYWORDS

rectangular selection, Unity 3D, WebGL, 3D Repo

Web3D '19, July 26–28, 2019, Los Angeles, CA, USA
2019. ACM ISBN 978-1-4503-6798-1/19/07.
https://doi.org/10.1145/3329714.3338125

ACM Reference Format:
Sebastian Friston, Jozef Doboš, Charence Wong, Carmen Fan, Santiago Montero, and Anthony Steed. 2019. Rectangular Selection of Components in Large 3D Models on the Web. In *Web3D '19: The 24th International Conference on 3D Web Technology (Web3D '19), July 26–28, 2019, Los Angeles, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3329714.3338125

1 INTRODUCTION

Even though the ability to render 3D scenes on the web has become commonplace due to the introduction of WebGL and the proliferation of open source libraries such as three.js [Cabello 2010], the number of online industrial 3D design tools is limited. Commercial systems such as *Onshape*, and *SketchUp for Web* provide online editors for mechanical Computer Aided Design (CAD) and architectural modeling respectively. Though these editors allow manipulation of individual objects in real time, the number of supported components is relatively low. SketchUp for Web, for instance, becomes unsuable with only 10, 000 cuboids being rendered in our tests.

In contrast, a full architecture, engineering and construction (AEC) design of a high-rise building such as the one shown in

3D Repo in Fig. 1 can consist of hundreds of thousands of components. Various techniques including but not limited to geometry simplification [Limper 2018; Limper et al. 2016], mesh batching [Scully et al. 2015] and streaming [Limper et al. 2014; Schilling et al. 2016; Scully et al. 2016] have been deployed to support complicated models on web browsers. Yet, despite all this added complexity and the limitations of WebGL, the users of such systems still expect functionality similar to desktop authoring tools. This includes object selection, showing and hiding of components, etc. Unfortunately, as large models become optimized and thus unavailable as primitives, even a simple task of rectangular selection, i.e. grouping and highlighting of objects en masse, becomes difficult as traditional techniques are no longer applicable.

A simple approach to rectangular selection would be to project a 2D screen-space rectangle into 3D space in order to calculate per-object intersections. However, this approach does not work with mesh batching, whereby the complexity of a scene graph is reduced by joining multiple primitives—known as *submeshes*—into larger structures called *supermeshes*.

An alternative would be to project objects into the screen space so that fragments can be tested against the user-defined selection rectangle. Even though this would be trivial on a modern graphics processing unit (GPU), the need to distinguish individual submeshes from within supermeshes would be computationally expensive in WebGL due to fragment shader limitations further explained in §3.

Thus, in this paper we set out to define a novel method that supports rectangular selection of objects, firstly by performing intersection tests against bounding boxes of submeshes and then by false-color rendering of those submeshes that have not yet been eliminated by the first test. This has been implemented in Unity WebGL as part of the 3D Repo design coordination platform [Friston et al. 2017]. Five AEC federated scenes consisting of architecture, structure and mechanical, electrical and plumbing models have been tested across a range of devices and different sizes of rectangular selections to ensure that the proposed solution works well in real-world deployment. This solution is freely available at https://3drepo.com

Contributions. In order to provide selection functionality akin to desktop authoring tools as shown in Fig. 1, we present a method for rectangular selection of components in large 3D models on the web. Hence, our contributions can be summarized as follows:

(1) Definition of a novel method for broad and narrowphase collision detection suitable for WebGL;
(2) Implementation in Unity game engine and deployment within the 3D Repo production environment;
(3) Experimental evaluation on a range of industrial 3D models varying in size and complexity across different devices and screen resolutions.

2 RELATED WORK

Our work builds on prior collision detection art while being specifically designed to support WebGL. The implementation has been developed within the 3D Repo platform online.

2.1 3D Repo

3D Repo [Doboš and Steed 2012] is an open source design coordination system specifically developed for the AEC industry. Over the years, the platform evolved from an early XML3D-based [Sons et al. 2010] visualization and data hosting prototype [Doboš et al. 2013] into a commercially viable offering *3drepo.io* [Scully et al. 2015] based on X3DOM [Behr et al. 2009] and later Unity 3D game engine [Friston et al. 2017]. The latest iteration includes 3D Diff [Doboš et al. 2018a] which detects real-time differences between any two 3D models regardless of their underlying file type directly on a web browser. An inverse calculation visualizes clashes, i.e. areas where objects collide, which is immesely useful in AEC, too. This technology has been applied to some of the largest and most prestigious construction projects worldwide with companies such as Atkins, Balfour Beatty, Canary Wharf Contractors, HOK and Skanska among others [Doboš et al. 2018b].

The core of the platform consists of three main systems, namely a processing library, a database and a web server. The 3D Repo Bouncer C++ library provides a heavy-lifting processing back-end which loads, decomposes and optimizes various industrial 3D encodings. As of 2019, the platform provides native support for proprietary file formats including Autodesk Revit and Bentley DGN. These are stored at the object level in polymorphic Binary JSON (BSON) collections within a NoSQL database, MongoDB. All of the data is processed and stored in 3D Repo's internal scene graph representation together with a revision history and highly optimized data blocks for eventual rendering on the web. A Node.js web server provides dynamic web pages and exposes a representational state transfer (REST) application programming interface (API) to the client listing user profiles, available projects, revision histories, comments, scene graphs, metadata, usage statistics, etc.

Even though 3D Repo itself does not provide any 3D editing functionality, its users are able to create new projects, upload revisions, create model federations, mark-up issues, perform data analytics and sensor integrations as well as record health and safety hazards within. The mark-ups follow the BIM Collaboration Format (BCF) [buildingSMART 2016] schema which is effectively an XML-based encoding for exchange of user-defined camera views, screenshots and textual annotations between AEC applications.

Similar web platforms with support for industrial size 3D models include solutions such as Autodesk 360 based on Autodesk Large Model Viewer which in turn is based on three.js, Bentley Web Navigator based on CesiumJS, and Instant 3D Hub based on a commercial version of X3DOM [Behr et al. 2015; Mouton et al. 2014], c.f. [Behr et al. 2018].

2.2 Collision Detection

Object selection can be seen as a special case of discrete collision detection. Collision detection typically operates in two stages. First, a filtering stage enumerates potentially colliding primitives (*broad-phase*) with fast but inexact tests, after which slower but completely robust intersection tests are performed (*narrowphase*) [Weller 2013].

Ericsson provides a comprehensive review of collision detection [Ericson 2004]. There is much recent work on applying GPUs to collision detection utilising the latest shader model features or frameworks such as CUDA, often concerned with the application of acceleration structures, c.f. [Chitalu et al. 2018; Dos Santos et al. 2014; Vinkler et al. 2016]. However, the application of GPUs to collision detection began before introduction of general purpose compute features.

Typically, it was the broadphase stage that was accelerated, as the collision matrix maps well to the structured access patterns of the GPU [Weller 2013]. Govindaraju et al. [2003] and Jang & Han [2008] presented GPU based broadphase implementations, encoding results to the frame buffer. Later stages have also been implemented however. For example, Faure et al. [2008] and Rodriguez-Navarro et al. [2005] use depth images to perform per-object intersection tests.

Where our problem differs from most previous work is in its batching implementation. AEC models typically have large numbers of small objects that can saturate the CPU with draw calls. Our implementation combines these objects into supermeshes. When rendered, shaders look up object-specific parameters from textures on a per-fragment basis. The disadvantage for collision detection is that the CPU no longer has access to sub-object representations (e.g. triangles) on which traditional tests can be performed. Our implementation is most similar to the image-space tests of Faure et al. and Rodriguez-Navarro et al.

3 OBJECT SELECTION

Object selection is a special application of an intersection test(s). The most straightforward solution is to test each primitive against a selection volume or region individually and perform a logical `OR` on the results.

There are two ways to do this. The selection region is projected into primitive space (e.g. a rectangle would become a frustum), and traditional intersection tests are performed against this volume. Such tests can be optimised with acceleration structures such as a bounding volume hierarchy (BVH) [Dinas and Bañón 2015]. This is simple, and potentially very fast with an efficient acceleration structure, even though it does have some disadvantages. If a structure is not available, one must be computed at possibly high cost. While rectangles and circles project to frustums and cones for which efficient intersection tests are readily available [Ericson 2004], this is not true of all shapes. Crucially though, this approach does not support submeshes, because the geometric tests are performed with individual primitives.

Alternatively, primitives can be projected into screen space, and fragments tested against a region so only the visual geometry is required. Unfortunately, not all 2D intersection tests are equally efficient. Odd shapes can be better supported with, for example, rasterized stencils. Submeshes are supported, but with a caveat; it is not sufficient only to rasterize, but each fragment must be shaded/evaluated in order to determine its subobject ID. This is effectively a software rasterization, and for large models, will be highly inefficient.

Ideally, we would leverage the GPU to achieve performance equivalent to a color render. As the GPU is already configured for rasterization, the problem is how to communicate the intersection tests to the CPU.

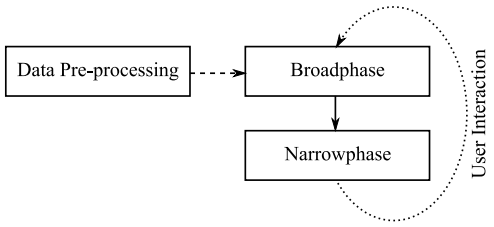


Figure 2: Implementation diagram: Model data is pre-processed on load and then object collisions are calculated fistly in broadphase and then narrowphase to determine user selection. These phases are repeated each time user draws a 2D selection rectangle across the screen.

On modern GPUs this is trivial, because in Shader Model 5 (SM5) fragment shaders support unordered (random) write access. Therefore each fragment could perform an intersection test and set flags in a buffer of object IDs. For now, however, WebGL supports SM3, which provides only structured write accesses [Khronos Group 2019]. That is, the location in a texture to which a result is written, and the location on the screen that is tested against the region, are inextricably linked. This prevents the traditional use of flags, and is the crux of the problem that must be overcome to perform sub-object selection in screen space with WebGL.

4 IMPLEMENTATION

Our implementation uses both techniques from §3 to perform GPU accelerated object selection as shown in Fig. 2. First, mesh data is pre-processed before rendering. Next, a broadphase test allows early acceptance or rejection of submeshes based on axis-aligned bounding boxes (AABBs). Finally, in the second pass, ambiguous submeshes are tested one by one in image space.

4.1 Data Pre-processing

The selection tests require some pre-processing of data. However, the data is simple and used predominantly for bookkeeping, so can be computed in linear time with respect to the number of submeshes causing minimal overheads, and only once.

- Submesh AABBs in world space are required. These are pre-computed when the submeshes are packed into supermeshes.
- Globally unique identifier (GUID) is required for each submesh. These are computed when the supermeshes are loaded for the first time.
- Supermesh lookups for each submesh GUID are created for the purposes of rendering by submesh.

4.2 Broadphase

The broadphase, shown in Fig. 3, is performed against a linear array of submesh bounding boxes in world space computed in a pre-processing pass. Each submesh is assigned a globally (process-wide) unique index. A set of AABBs are computed server side and delivered along with the supermeshes.

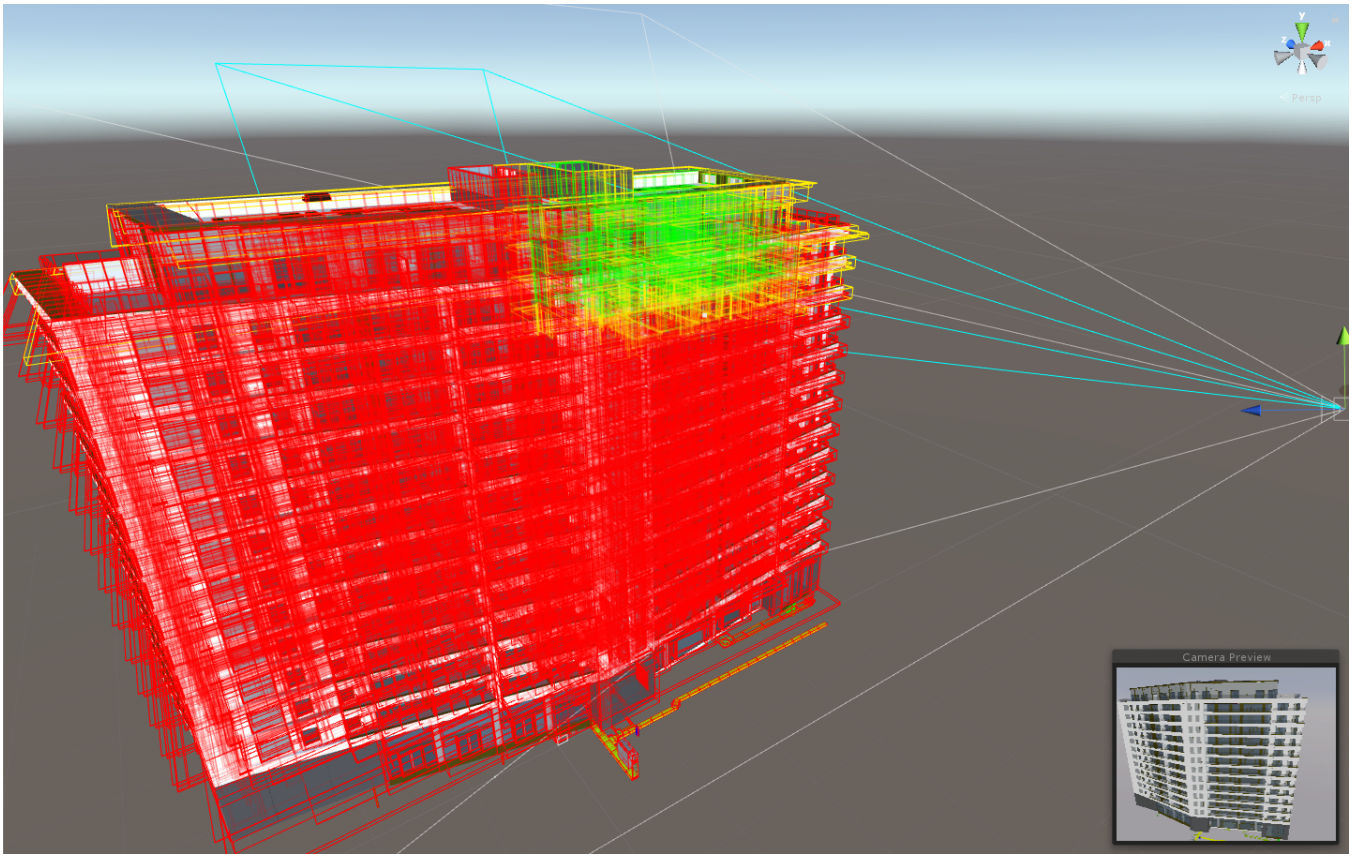


Figure 3: Visualization of the broadphase test. The selection rectangle is projected into a frustum against which AABBs are tested. This model (A2) consists of over 100k components. Model courtesy of Canary Wharf Contractors.

For the GPU accelerated broadphase, the bounding boxes are encoded in a set of meshes, with the vertices defined as:

$$\begin{aligned} vertex &= bounds.center; \\ uv0.x &= bounds.extents.x; \\ uv0.y &= bounds.extents.y; \\ uv1.x &= bounds.extents.z; \\ uv1.y &= 0f; \\ uv2.x &= globalindex; \\ uv2.y &= submeshindex; \end{aligned}$$

The *globalindex* is used to compute a location in the render-texture into which to write the result. The location determines to which submesh the result pertains and the color the result of the test. Mapping between submesh GUID and location is given by:

$$f(guid, mapsize) = [\lfloor \frac{guid}{mapsize_y} \rfloor, guid \mod mapsize_y] + 0.5$$

The half-texel offset is to force sampling the centre of the pixel on the GPU.

In practice, the map is always square, being given by:

$$mapsize(num_{guid}) = [\lceil \sqrt{num_{guid}} \rceil, \lceil \sqrt{num_{guid}} \rceil]$$

The intersection test is performed against a frustum in world space consisting of six planes. The planes are computed by unprojecting the corners of the rectangle from screen space into world space, for both the near and far plane, and determining the plane parameters from three vertices on each of the sides.

Assarsson & Moller [1999] describe the basic AABB-Frustum test. Each AABB is tested against each plane. If a box is entirely outside any plane, it cannot intersect the frustum. If the box is entirely inside all planes, it is considered within the frustum, otherwise it is intersecting. Assarsson & Moller describe a number of optimisations, though the simplest test is performant enough with GPU acceleration.

The intersection test, therefore, has three possible outcomes, two of which allow us to conclude the state of the submesh without performing any screen space tests. The remaining submeshes are forwarded to the narrowphase.

4.3 Narrowphase

During the narrowphase, each submesh is rendered with a false-color pass on its own. This is done by rendering the supermesh and discarding any fragments not belonging to the desired submesh.

Each fragment tests itself against the selection region and encodes whether it is inside or outside using the Red and Green channels. The render texture is then reduced using a prefix-or operation. Over $log_2(max(width, height), Kernel)$ steps, new textures are computed by subsampling the previous texture, where each subsample color is a the element-wise maximum of all the colors in a *kernel* size square.

Non power-of-two textures are handled by setting the sampler to clamp at the edges and starting with an oversize texture. Point filtering is used, however the nature of the encoding (each channel is a flag) and the prefix operation (max) should mean this should not affect the result, only the execution time. The final result of the iterative subsampling is a 1×1 texture with the color channels as flags encoding the result of the screen space test.

To reduce GPU synchronizations, the results are copied from each subsampled texture into an accumulator. This is done by rewriting the accumulator texture for each copy, but with clear flags disabled. On each write, the fragment shader determines (based on location) if the pixel pertains to the flags currently in the 1×1 source texture. If so that pixel is written, otherwise it is left as is.

Finally, the accumulator texture is read back from the GPU, parsed and the results integrated with those from the broadphase.

5 USER INTERFACE

Based on the requirements of the AEC professionals working with 3D Repo, the rectangular selection tool has been implemented to provide multiple modes of interaction similar to popular desktop 2D drawing and 3D modeling applications. On WebGL Unity, the rectangular selection is enabled by pressing and holding a `Shift` key which adds an icon modifier to the mouse cursor for visual feedback as shown in Fig. 4. The user then drags the mouse across the screen to draw a rectangle with color overlay.

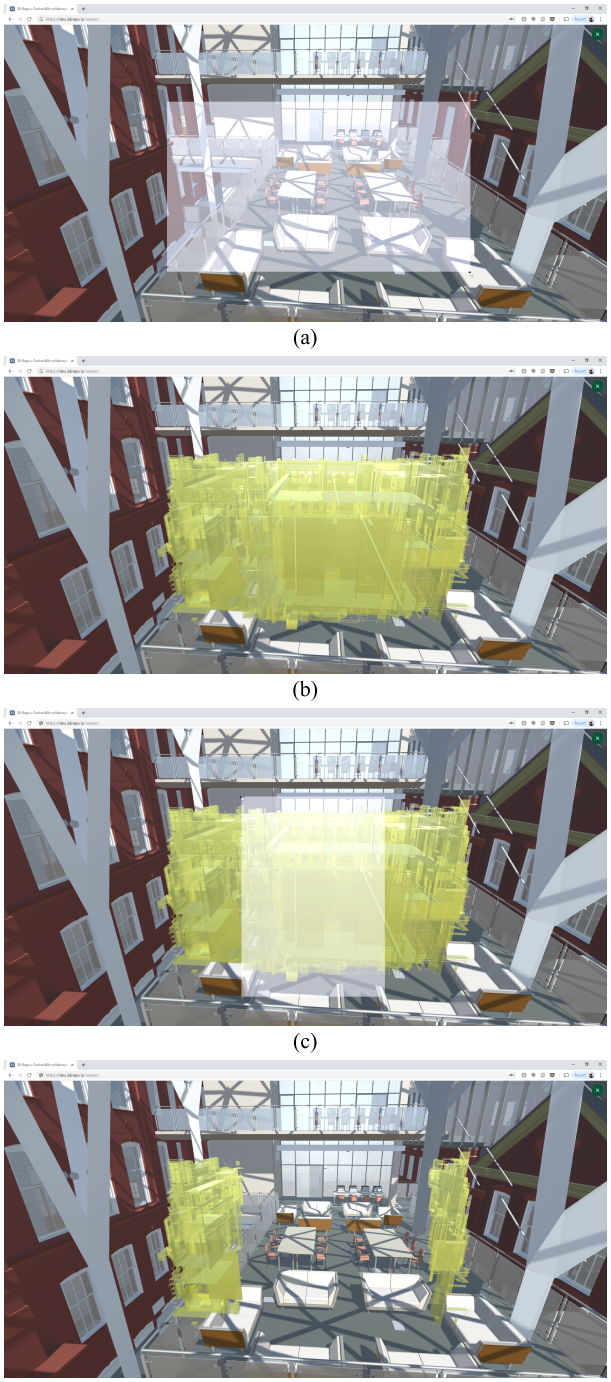
The supported modes are:

Select any. Dragging the rectangle from the top left to the bottom right selects all objects that are at least partially contained within the rectangle.

Select contained only. Dragging the rectangle in the opposite direction, i.e. from the bottom right to the top left, selects only those objects that are fully contained within the rectangle. Any partial containement is ignored.

Multi-select/deselect. Either of the previous selection modes can be further modified to enable addition (`Ctrl`) or removal (`Alt`) of objects in respect of the existing selection. This further enhances the usefulness of the tool so that users can perform basic boolean operations on vast numbers of objects easily.

A single narrowphase detects whether an object was encountered both inside and outside of the selection rectangle, and returns the results in separate flags. The combination of these flags allows the CPU to determine whether an object was partially or wholly selected, and select or deselect it depending on the mode and existing selection state. The performance is therefore equivalent for all selection modes. Once the objects are selected, they are displayed with an X-ray higlight so that the selection stands out from the rest of the model. This is a final pass shader effect which renders the selected objects on top of the scene with depth test disabled.



(a) (b) (c) (d)

Figure 4: An example of selection and deselection. a) The user draws a large rectangle across the screen. b) The system selects all components that fall within the selected volume. c) The user draws a smaller rectangle in the opposite direction with `Alt` modifier enabled. d) The system deselects all components that fall at least partially within the selected volume. Model courtesy of Bond Bryan Digital.

6 EVALUATION

To evaluate the proposed solution, we have tested the Unity-based implementation with five different industrial AEC models, shown in Fig. 5, across a range of different devices. Each scene was loaded automatically in the default 45° isometric perspective rendering showing the full extent of the model on screen. The selection rectangle was then invoked progressively covering 1/1, 1/2, 1/4, 1/8 and 1/16 of the screen from the the centre outwards as shown in Fig. 6. This ensured the test being repeatable covering the same number of objects regardless of screen resolution.

Fig. 5 shows the respective AEC test scenes that themselves consist of multiple individual models composed into co-located federations. These scenes have been carefully selected to represent real-world engineering models with a significant number of components (submeshes) that have been optimized by joining the rendering graph elements into supermeshes.

Tab. 1 shows the total execution times of the selection tests for the models. The tests performed are the same regardless of selection mode. As can be seen approximately 75% of the tests show acceptable response times for users of AEC visualization tools, which is informally but commonly understood as a response time under 5 seconds. In extreme cases – large models on low power devices – the tests are unreasonably slow. Due to the high redundancies in the narrowphase, our method is sensitive to suboptimal fill rates on low power devices. However this is not unique to our method and such devices struggle regardless, with frame rates too low to be navigable (5 fps) even without our selection tool execution.

Fig. 7 shows the narrowphase/broadphase breakdown for each test. As can be seen the execution times are dominated completely by the narrowphase regardless of platform.

Fig. 8 shows the narrowphase/broadphase breakdown by resolution. Both tests are dependent on resolution to an extent. The broadphase does not perform any image space tests, and so the dependency on resolution will likely be due to the relative size and positioning of the selection rectangle with respect to the model. We would expect the narrowphase to be highly dependent, being an image space test, though the correlation is actually quite weak, and can be explained by the efficiency variation of the broadphase, given the correlation strength. These results are far from conclusive, but suggest that the execution times are less reliant on the GPU throughput, than the CPUs ability to dispatch draw calls, since the former will be directly affected by resolution, but the latter only weakly (dependent only on the number of reduction stages).

7 DISCUSSION

The narrowphase stage is very inefficient per-test result, but it is important to remember the majority of the tests are performed in the broadphase. For example, the test shown in Figure 3 considers 106,401 objects, of which 98,721 are culled, 6,889 immediately selected, leaving 791, or 0.74%, of objects to proceed to the more costly narrowphase. Across all the tests the broadphase was $\overline{98\%}$ efficient ($\sigma = 2\%$). This percentage is not so much dependent on the size of the selection rectangle, which only alters the ratio of immediate accepts or rejects, but rather the distribution of sub-objects and their bounding primitive efficiency. Thus, the real performance indicator is the narrowphase, which depends on the number of submeshes

Table 1: Timing results (in seconds) of the experimental evaluation defined in §6. Five different models with decreasing complexity have been tested across three different GPUs and screen resolutions. Corresponding models are further depicted in Fig. 5.

		A1	A2	A3	Heartspace	Skanska
Federated models		9	4	2	9	9
Supermeshes		54	37	16	31	33
Submeshes		335,477	106,401	74,966	51,464	47,199
Intel HD Graphics 520 1920x1080px	**1/16**	20.567	0.749	1.349	10.262	5.507
	1/8	37.043	0.701	0.940	8.327	1.358
	1/4	34.884	0.377	0.849	7.685	1.410
	1/2	17.184	0.095	0.535	3.848	1.489
	1/1	0.984	0.076	0.196	0.591	1.465
Nvidia GTX 1050 3840x2160px	**1/16**	25.768	0.835	0.120	26.880	20.666
	1/8	27.462	0.187	0.039	28.085	31.385
	1/4	28.965	0.214	0.064	29.549	32.479
	1/2	4.344	0.212	0.315	4.201	15.583
	1/1	1.376	0.213	0.959	1.363	0.694
Nvidia GTX Titan 4096x2160px	**1/16**	3.461	9.507	3.461	4.593	5.498
	1/8	3.159	7.882	3.159	1.518	4.726
	1/4	3.599	8.082	3.599	1.625	2.400
	1/2	1.037	4.032	1.037	1.704	0.779
	1/1	0.692	0.687	0.692	1.654	0.514

that intersect the rectangle edges, so selecting the entire model will be almost equally as fast on any model as it depends on just a single draw call.

In our problem of AEC visualization, the distribution of submeshes is safely assumed, though this may not be the case for all domains. With physically implausible distributions, such as with artist per-material batching, or stylized content, the broadphase efficiency will decrease leading to an outsized increase in execution time. Higher resolution models will further inhibit the narrowphase throughput. However, domains such as entertainment do not typically model individual components and so have lower object counts. They could therefore have better performance due to fewer total narrowphase passes, even with reduced broadphase efficiency.

As visible in both Tab. 1 and Fig. 7, it takes far longer to compute the smallest selection rectangle which might be a counter-intuitive result at first. However, this is due to a smaller rectangle culling proportionally less components using broadphase early accept/reject calculation. Since each dimension of the rectangle has been halved at each step, the time increases proportionally. This resolution-depdendent trendline for both the broad- and narrowphase is shown in Fig. 8. It should be noted that the higher selection times between scenes A1 and A2 at different selection sizes are due to the distribution of components across the screens. A1 is a tall and narrow building while A2 covers more of the overall space.

Nevertheless, the major negative result are the very slow response times on both the Intel HD 520 chip and NVidia GTX 1050 graphics card across the most complex models. However, NVidia was rendering four times many more pixels than Intel which explains the very similar results between them. Also, such complex models achieve sub five frames per second rendering which is considered non-interactive even without the additional overhead of

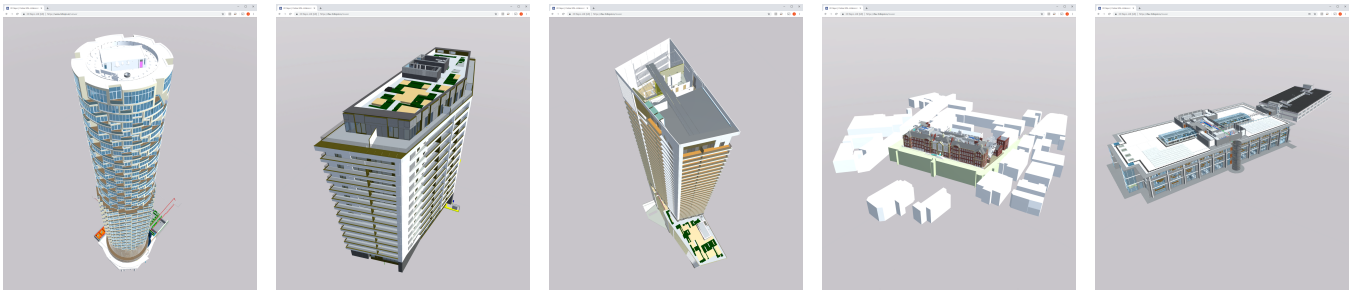


Figure 5: Complex AEC scenes used in our experimental evaluation in §6. Models courtesy of Canary Wharf Contractors, Bond Bryan Digital and Skanska respectively.

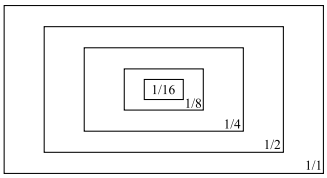


Figure 6: Progressive sizing of the selection rectangle used in the experimental evaluation.

selection rendering. Thus, these results are included only for information purposes and do not affect the real-world performance of our technique.

7.1 Interactive Selection

Even though it is possible to display object highlights and component counts as the user drags a rectangular selector across the screen in real-time, it is not advisable since the computational overhead on lower-end devices might cause a noticeable delay. Thus, our implementation performs the broad- and narrowphase collision detections only once when the full rectangle is drawn and the user lifts the mouse button.

8 FUTURE WORKS

Optimizations. Each narrowphase test has three possible results, and so requires only 2 bits, whereas the frame buffer to which the results are written is up to 32-bit. Ideally then, multiple narrowphase tests could be batched and performed with a single draw-call & subdivision. This could be achieved by disabling the depth buffer checks and performing a bitwise `OR`, both in the blend operation and texture reduction shader. This could offer a draw call reduction of ×16, however requires the use of bitwise operations that are only available in SM4. A variation of this could use the max operations which are available in SM3. These operations run on each color channel separately.

Therefore, if we found an encoding that allowed a max operation to behave, for a single test result, as a semantic `OR`, we could still reduce the draw call count by ×4. Based on the predominance of the narrowphase to the total execution time, this could be highly impactful, reducing the most egregious test result from 25 seconds to 6. Even if we were unable to devise an encoding for a 2-bit result per channel, and instead had to just run the narrowphase twice, the workload could still be reduced by a factor of ×2 with all four channels in use per test.

Rubber-band selection. An advantage of image-space based tests is that complex, non-parametric, bounding shapes can be supported very efficiently. For example, through the use of a rasterization into a stencil. This would severely impact the broadphase efficiency however, as it would no longer be possible to perform early accepts.

AEC Deltas. The ability to sub-select components en masse is the basis of the AEC Deltas project (https://github.com/aecdeltas) which aims to group and transmit delta changes between 3D Repo, desktop-based authoring tools, Speckle Works by Arup and the Buildings and Habitats object Model (BHoM) library by Buro Happold Engineering.

9 CONCLUSION

This paper presented a GPU-based method for rectangular selection on web-based rendering systems that rely on mesh batching. Our implementation was based on 3D Repo and combined early reject & accept intersection testing in the world space with fine-grained final intersection testing in the screen space. We evaluated the performance of our technique with five different AEC industrial 3D models ranging from tens to hundreds of thousands of components. While some combinations of lower power device and large model had unteneable execution times, the majority of response times were in the ranges expected by a typical user of such visualization packages. Thus, this solution has been deployed in a production environment with thousands of users worldwide.

ACKNOWLEDGMENTS

This work has been kindly supported by Innovate UK Increase Productivity, Performance and Quality in UK Construction grant No. 104799 as well as Horizon 2020 grant No. 700294.

REFERENCES

Ulf Assarsson and Tomas Moller. 1999. Optimized View Frustum Culling Algorithms. *Jgt* 5, 1 (1999), 29. https://doi.org/10.1080/10867651.2000.10487517

Johannes Behr, Peter Eschler, Yvonne Jung, and Michael Zöllner. 2009. X3DOM: A DOM-based HTML5/X3D Integration Model. In *Proceedings of the 14th International Conference on 3D Web Technology (Web3D '09)*. ACM, New York, NY, USA, 127–135. https://doi.org/10.1145/1559764.1559784

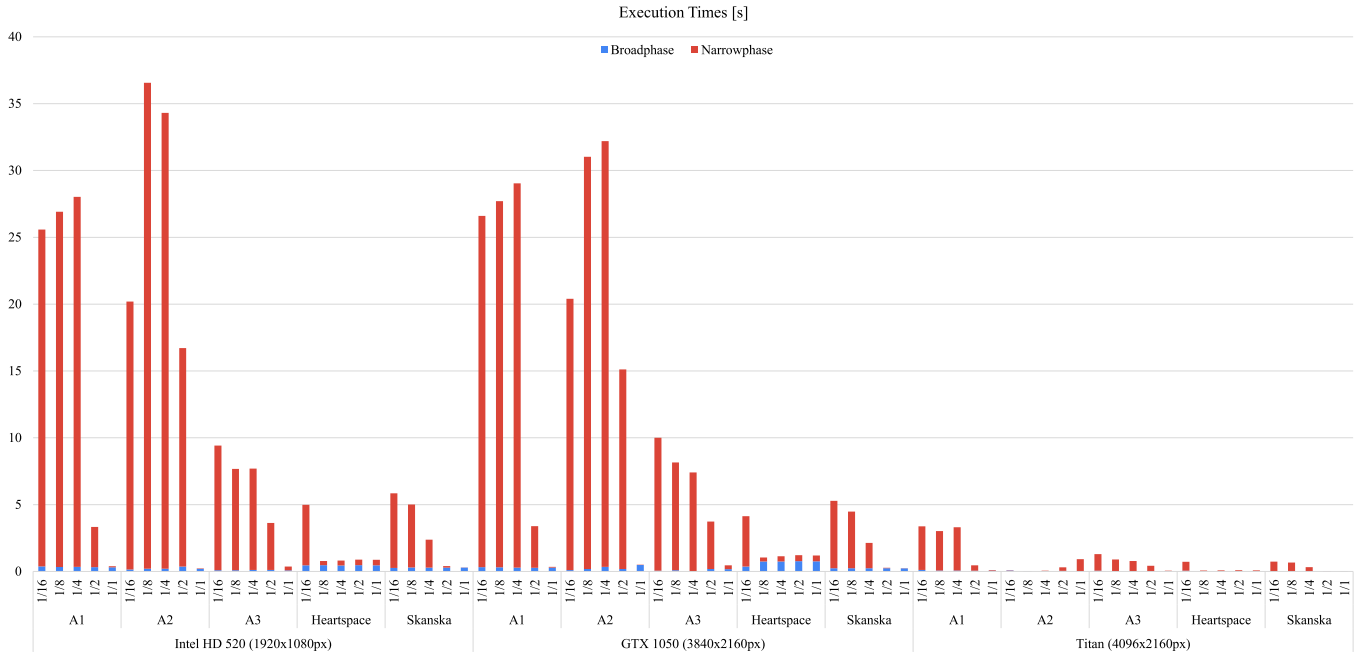


Figure 7: Execution times (in seconds) split by narrowphase & broadphase for each test listed by GPU.

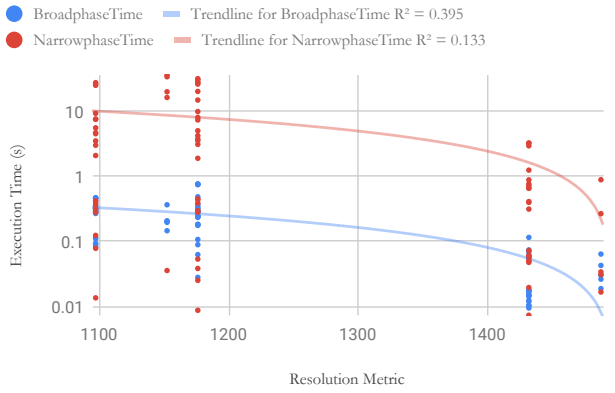


Figure 8: Execution times (log scale) split by narrowphase & broadphase for each against resolution metric (the diagonal size).

Johannes Behr, Max Limper, and Timo Sturm. 2018. MoST: A 3D Web Architectural Style for Hybrid Model Data. In *Proceedings of the 23rd International ACM Conference on 3D Web Technology (Web3D '18)*. ACM, New York, NY, USA, Article 21, 8 pages. https://doi.org/10.1145/3208806.3208823

Johannes Behr, Christophe Mouton, Samuel Parfouru, Julien Champeau, Clotilde Jeulin, Maik Thöner, Christian Stein, Michael Schmitt, Max Limper, Miguel de Sousa, Tobias Alexander Franke, and Gerrit Voss. 2015. webVis/Instant3DHub: Visual Computing As a Service Infrastructure to Deliver Adaptive, Secure and Scalable User Centric Data Visualisation. In *Proceedings of the 20th International Conference on 3D Web Technology (Web3D '15)*. ACM, New York, NY, USA, 39–47. https://doi.org/10.1145/2775292.2775299

buildingSMART. 2016. Web service specification for BIM Collaboration Format. https://github.com/BuildingSMART/BCF-API.

Ricardo Cabello. 2010. Three.js JavaScript 3D library. https://github.com/mrdoob/three.js/.

Floyd M. Chitalu, Christophe Dubach, and Taku Komura. 2018. Bulk-synchronous parallel simultaneous BVH traversal for collision detection on GPUs. *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D '18* (2018), 1–9. https://doi.org/10.1145/3190834.3190848

Simena Dinas and José M Bañón. 2015. A literature review of bounding volumes hierarchy focused on collision detection. *Revista Ingeniería y Competitividad* 17, 1 (2015), 49–62.

Jozef Doboš, Carmen Fan, Sebastian Friston, and Charence Wong. 2018a. Screen Space 3D Diff: A Fast and Reliable Method for Real-time 3D Differencing on the Web. In *Proceedings of the 23rd International ACM Conference on 3D Web Technology (Web3D '18)*. ACM, New York, NY, USA, Article 9, 9 pages. https://doi.org/10.1145/3208806.3208809

Jozef Doboš, Carmen Fan, Pavol Knapo, and Charence Wong. 2018b. Applications of Web3D Technology in Architecture, Engineering and Construction. In *Proceedings of the 23rd International ACM Conference on 3D Web Technology (Web3D '18)*. ACM, New York, NY, USA, Article 30, 2 pages. https://doi.org/10.1145/3208806.3219741

Jozef Doboš, Kristian Sons, Dmitri Rubinstein, Philipp Slusallek, and Anthony Steed. 2013. XML3DRepo: A REST API for Version Controlled 3D Assets on the Web. In *Proceedings of the 18th International Conference on 3D Web Technology (Web3D '13)*. ACM, New York, NY, USA, 47–55. https://doi.org/10.1145/2466533.2466537

Jozef Doboš and Anthony Steed. 2012. 3D Revision Control Framework. In *Proceedings of the 17th International Conference on 3D Web Technology (Web3D '12)*. ACM, New York, NY, USA, 121–129. https://doi.org/10.1145/2338714.2338736

A.L. Dos Santos, V. Teichrieb, and J. Lindoso. 2014. Review and comparative study of ray traversal algorithms on a modern GPU architecture. *22nd International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, WSCG 2014, Full Papers Proceedings - in co-operation with EUROGRAPHICS Association* June (2014), 203–212.

Christer Ericson. 2004. *Real-Time Collision Detection*. CRC Press.

François Faure, Sébastien Barbier, Jérémie Allard, and Florent Falipou. 2008. Image-based Collision Detection and Response between Arbitrary Volumetric Objects. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Dublin, Ireland, 1–8. http://dl.acm.org/citation.cfm?id=1632615

Sebastian Friston, Carmen Fan, Jozef Doboš, Timothy Scully, and Anthony Steed. 2017. 3DRepo4Unity: Dynamic Loading of Version Controlled 3D Assets into the Unity Game Engine. In *Proceedings of the 22Nd International Conference on 3D Web Technology (Web3D '17)*. ACM, New York, NY, USA, Article 15, 9 pages. https://doi.org/10.1145/3055624.3075941

NK Govindaraju and Stephane Redon. 2003. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. *Acm Siggraph/Eurographics* (2003), 25–32. http://dl.acm.org/citation.cfm?id=844178

Hanyoung Jang and Junghyun Han. 2008. Fast collision detection using the A-buffer. *The Visual Computer* 24, 7-9 (jul 2008), 659–667. https://doi.org/10.1007/s00371-008-0246-8

Khronos Group. 2019. WebGL 2.0 Specification. https://www.khronos.org/registry/webgl/specs/latest/2.0.

Max Limper. 2018. *Automatic Optimization of 3D Mesh Data for Real-Time Online Presentation.* Ph.D. Dissertation. Technische Universität, Darmstadt. http://tuprints.ulb.tu-darmstadt.de/7500/

M. Limper, A. Kuijper, and D. W. Fellner. 2016. Mesh Saliency Analysis via Local Curvature Entropy. In *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics: Short Papers (EG '16)*. Eurographics Association, Goslar Germany, Germany, 13–16. https://doi.org/10.2312/egsh.20161003

Max Limper, Maik Thöner, Johannes Behr, and Dieter W Fellner. 2014. SRC - a streamable format for generalized web-based 3D data transmission. In *Proceedings of the 19th International Conference on 3D Web Technology (Web3D '14)*. ACM, New York, NY, USA, 35–43. https://doi.org/10.1145/2628588.2628589

Christophe Mouton, Samuel Parfouru, Clotilde Jeulin, Cecile Dutertre, Jean-Louis Goblet, Thomas Paviot, Samir Lamouri, Max Limper, Christian Stein, Johannes Behr, and Yvonne Jung. 2014. Enhancing the Plant Layout Design Process Using X3DOM and a Scalable Web3D Service Architecture. In *Proceedings of the 19th International ACM Conference on 3D Web Technologies (Web3D '14)*. ACM, New York, NY, USA, 125–132. https://doi.org/10.1145/2628588.2628592

J. Rodriguez-Navarro, M. Sainz, and A. Susin. 2005. GPU Based cloth simulation with Moving Humanoids. In *Proc. XV Congreso Español de Informática Gráfica (CEIG2005)*. 147–155. https://doi.org/10.1.1.331.6845

Arne Schilling, Jannes Bolling, and Claus Nagel. 2016. Using glTF for Streaming CityGML 3D City Models. In *Proceedings of the 21st International Conference on Web3D Technology (Web3D '16)*. ACM, New York, NY, USA, 109–116. https://doi.org/10.1145/2945292.2945312

Timothy Scully, Jozef Doboš, Timo Sturm, and Yvonne Jung. 2015. 3Drepo.Io: Building the Next Generation Web3D Repository with AngularJS and X3DOM. In *Proceedings of the 20th International Conference on 3D Web Technology (Web3D '15)*. ACM, New York, NY, USA, 235–243. https://doi.org/10.1145/2775292.2775312

Timothy Scully, Sebastian Friston, Carmen Fan, Jozef Doboš, and Anthony Steed. 2016. glTF Streaming from 3D Repo to X3DOM. In *Proceedings of the 21st International Conference on Web3D Technology (Web3D '16)*. ACM, New York, NY, USA, 7–15. https://doi.org/10.1145/2945292.2945297

Kristian Sons, Felix Klein, Dmitri Rubinstein, Sergiy Byelozyorov, and Philipp Slusallek. 2010. XML3D: Interactive 3D Graphics for the Web. In *Proceedings of the 15th International Conference on Web 3D Technology (Web3D '10)*. ACM, NY, USA, 175–184. https://doi.org/10.1145/1836049.1836076

Marek Vinkler, Vlastimil Havran, and JiÅŽí Bittner. 2016. Performance Comparison of Bounding Volume Hierarchies and Kd-Trees for GPU Ray Tracing. *Computer Graphics Forum* 35, 8 (dec 2016), 68–79. https://doi.org/10.1111/cgf.12776

René Weller. 2013. A Brief Overview of Collision Detection. In *New Geometric Data Structures for Collision Detection and Haptics*. Springer International Publishing, Heidelberg, 9–47. https://doi.org/10.1007/978-3-319-01020-5